A SVM-based Software Homology Detection Method

Bang Sun[1, 2*]**, Xiaoming Liu**[3*]**, Dian Lei**[1, 2]**, Qi Li**[1, 2]
[1]Beijing University of Posts and Telecommunications, Beijing, China
[2]NEL of Security Technology for Mobile Internet, Beijing, China
[3]National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China
475931153@qq.com, liuxm@cert.org.cn

Abstract: **With the development of software technology, software is becoming increasingly essential in our daily lives. However, there are more and more sorts of pirated software. Therefore, we need to figure out an effective method to prevent plagiarism, protect intellectual property and so on. Software homology detection can be used as protection of software intellectual property, malware defense, and can chase an important means of accountability and evidence. We did some experiments to evaluate the performance of our method .And the results showed that the proposed method can achieve satisfactory outcome.**

***Keywords: Software homology detection, SVM,* software technology**

I. INTRODUCTION

For programmers, writing code is similar to normal life. Whether to eat, rest or entertain, everyone has his own living habits. And everyone has his own coding style. Although coding style cannot identify each person as accurately as fingerprints, but it still provides an evidence which can help us to identify the author. [1]

A. Importance.

With the arrival of the Internet era, software homology detection is becoming more and more important. Software homology detection, in daily life can be used to detect the original student codes, check whether the students have the ability to program independently. Applied to business, the codes has their own copyright. So it is a better way to solve the dispute of software copyright. [2]

B. Difficulty.

First, obtaining the correct multiple samples is a difficult problem. Second, for an immature software developer, his habit of coding might change, which will lead to lower the accuracy rate .In the meantime, it will cost a lot of time training those selected effective coding features which are based on various programming languages.[3][4][5]

C. Contributions & Article structure.

In this paper, we propose a SVM-based method to detect the homology among different kinds of software. The proposed method is composed of two important stages: the training stage and the testing stage. In the training stage, we collect enough samples of each author and then train the models. In the testing stage, the test source code will be compared with each model. Then we decide whether the selected source code is composed by the author in our database. [6].

This paper is divided into six chapters, the main contents of each chapter are as follow: The first chapter is the introduction. The second chapter is an overview of the entire system. The third chapter is mainly about the feature selection process. The fourth chapter describes the SVM-based homology detection system .The fifth chapter describes the experiments. The sixth chapter gives the condition of this paper

II. SYSTEM OVERVIEW

As is shown in Fig. 1, the proposed homology detection system can be divided into two major parts--the offline training part and the online testing part.

In the offline training part, we first select five or more source code samples from one author followed by feature extraction and feature pre-treatment, and finally sends the processed samples to train SVM.[7][8]

In the online testing part, we randomly select a source code sample and then extract its features. Finally, we use our trained SVM model to test its similarity.

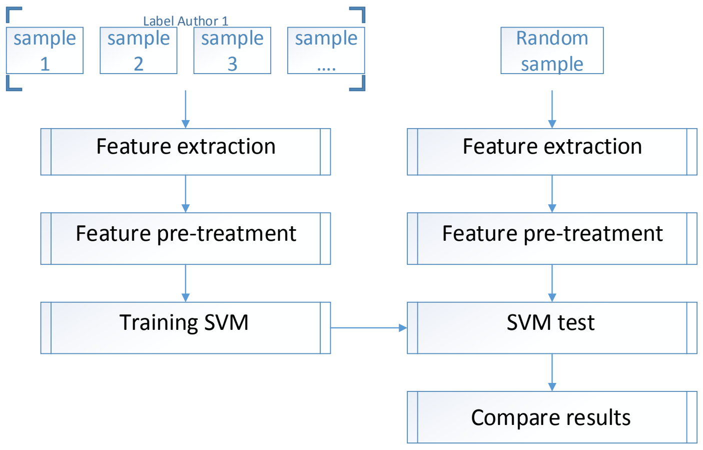


Fig. 1. Framework of the Source code detection

III. Feature Selection

A. *Basic Feature extraction*

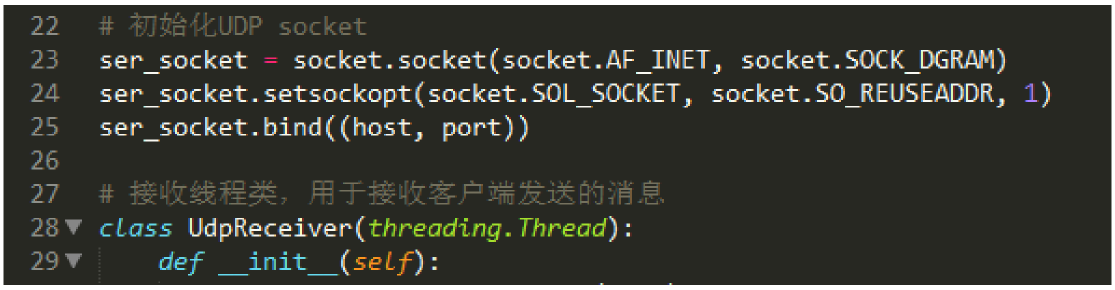
We have read a lot of source code, and after researching, collecting statistics, analyzing, finally we identified 12 characteristics to distinguish different authors' programming habits. For statistical purpose, we put the 12 characteristics into the vector group. As is shown in Table 1. [9]

TABLE I. Feature vector

Features	**description**	**Value vector**
F1	Code using the tab indents	1 0 0 0 0 0 0 0 0 0 0 0
F2	Code using the space indents	0 1 0 0 0 0 0 0 0 0 0 0
F3	A space before and after the Parameter in Parameter List	0 0 1 0 0 0 0 0 0 0 0 0
F4	Not a space before and after Parameter in Parameter List	0 0 0 1 0 0 0 0 0 0 0 0
F5	A space before and after the Operators	0 0 0 0 1 0 0 0 0 0 0 0
F6	Not a space before and after the Operators	0 0 0 0 0 1 0 0 0 0 0 0
F7	Variable names using camel	0 0 0 0 0 0 1 0 0 0 0 0
F8	Variable name style is not uniform	0 0 0 0 0 0 0 1 0 0 0 0
F9	Define function braces start a new line	0 0 0 0 0 0 0 0 1 0 0 0
F10	Define function braces do not start a new line	0 0 0 0 0 0 0 0 0 1 0 0
F11	Strings using single quotation marks	0 0 0 0 0 0 0 0 0 0 1 0
F12	Strings using double quotation marks	0 0 0 0 0 0 0 0 0 0 0 1

Take the follow source codes in fig.2 for example, the line- value vector of the selected code features are shown as follow:

$$Feature\_line_{21} = (0,0,0,0,0,0,0,0,0,0,0,0)$$
$$Feature\_line_{22} = (0,0,1,0,1,0,0,1,1,0,0,0)$$
$$Feature\_line_{24} = (0,0,1,0,1,0,0,1,1,0,0,0)$$
$$Feature\_line_{25} = (0,0,1,0,1,0,0,1,1,0,0,0)$$
$$Feature\_line_{26} = (0,0,0,0,0,0,0,0,0,0,0,0)$$
$$Feature\_line_{27} = (0,0,0,0,0,0,0,0,0,0,0,0)$$
$$Feature\_line_{28} = (0,0,1,0,1,0,1,0,1,0,0,0)$$
$$Feature\_line_{29} = (0,0,1,0,1,0,1,0,1,0,0,0)$$



```
# 初始化UDP socket
ser_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
ser_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
ser_socket.bind((host, port))

# 接收线程类，用于接收客户端发送的消息
class UdpReceiver(threading.Thread):
    def __init__(self):
```

Fig. 2. Code Sample

B. *Feature vector generation process*

As shown in Fig. 3.The feature vector is defined as $V = [V_1, V_2, \ldots, V_{12}]$. The extraction process is shown is fig.3.

We define $T_i = the\ number\ of\ lines\ for\ which\ F_i = 1$, $and\ FC_m = T_{2m-1} + T_{2m}$, so the $V_k = T_k / FC_{\lceil \frac{k}{2} \rceil}$

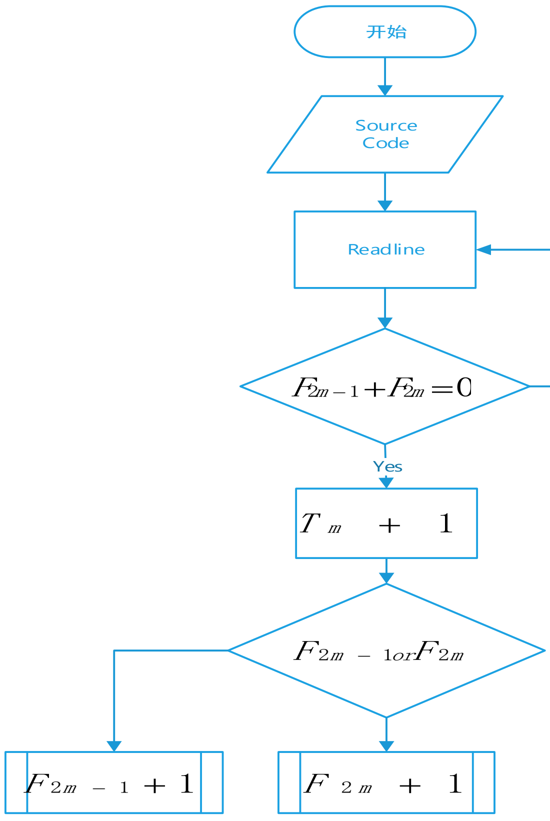


Fig. 3. Feature Extraction Process

IV. SVM(Support Vector Machine Theroy)

Vapnik proposed support vector machine (Support Vector Machine, SVM) algorithm in 1995. Optimization constraints are training errors. The optimization objective is to minimize the value of the confidence interval. SVM feature is based on the structural risk minimization criterion of learning and generalization ability was better than some of the traditional learning methods. SVM solving problems will ultimately be converted into solving quadratic programming problems. Therefore SVM solution is globally unique optimal solution. The accuracy of SVM classification algorithm have been well verified in many areas. SVM exhibits many unique advantages in solving the small sample, nonlinear and high dimensional pattern recognition problems. And it can be able to promote the use of machine learning to other problems such as function fitting. Classification and regression problems is attributed to solving convex quadratic optimization problem in SVM. From mathematical programming and optimization theory point view that secondary convex optimization problem existence theory and algorithm are study and solve problems thoroughly. This paper chose attack detection algorithm based on support vector machine (SVM) in. By support vector machine theory, can form a strict limitation of the optimal hyperplane to separate data set point in this super-flat sides, the principle shown in Fig 4.This theory for classification and identification of the characteristics of the attack detection process provides a theoretical basis.

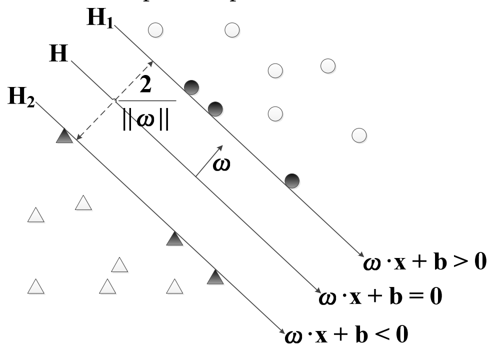


Fig. 4. Hyperplane schematic of SVM

V. Experiments

We selected codes from CSDN and codes written by ourselves .There are 140 kinds of codes from 14 authors, including C ++, JAVA, Python, and other programming languages The 12 features we selected are object-oriented language the common feature, so these languages can extract feature correctly. Respectively 14 authors' codes, after feature extraction, pretreatment, that are characterized by their code processed into a vector format and finally, training SVM. Then we randomly conducted three tests, with each test selected 30 code results in the following table.

It's clear that for most of the data, accuracy after using SVM test is in line with expectations, but still a small part of the data can't be identified correctly. Therefore, validation of the experiment, although the results of the experiment are not yet ideal. The selected 12 feature in this study is entirely feasible

TABLE II. Experiment Results

	Test 1	**Test 2**	**Test 3**
source	9 codes from author 6 6 codes from author 10 4 codes from author 5 7 codes from author 13 4 codes from author 2	7 codes from author 12 5 codes from author 14 8 codes from author 1 10 codes from author 3	3 codes from author 4 5 codes from author 11 8 codes from author 7 6 codes from author 9 8 codes from author 8
Accuracy	73.3%	86.7%	83.3%

VI. Conclusion

On the basis of researches existing software homology detection technology we developed a method of homology detection based on programming habits. Basing on our experiences and researches, we extracted 12 programming features that are suitable for various languages. In the research process, the codes of each author were disposed by feature extraction and SVM training. Use SVM powerful pattern recognition capabilities [12] detect software homology. It will provide effective help to the malware forensics (Author tracking), and copyright disputes solving [13]

Acknowledgment

This work is supported by National Natural Science Foundation of China Project (61302087, 61401038, U153610079).

References

[1] S. Afroz, M. Brennan, and R. Greenstadt. Detecting hoaxes, frauds, and deception in writing style online. In Security and Privacy (SP), 2012 IEEE Symposium on, pages 461–475. IEEE, 2012.

[2] A. Abbasi and H. Chen. Writeprints: A stylometric approach to identity-level identification and similarity detection in cyberspace. ACM Trans. Inf. Syst., 26(2): 1–29, 2008.

[3] N. Rosenblum, X. Zhu, and B. Miller. Who wrote this code? Identifying the authors of program binaries. Computer Security–ESORICS 2011, pages 172–189, 2011.

[4] VAPNIK V. The nature of statistical learning [M]. Berlin: Springer, 1995.

[5] F. Yamaguchi, C. Wressnegger, H. Gascon, and K. Rieck. Chucky: Exposing missing checks in source code for vulnerability discovery. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pages 499–510. ACM, 2013.

[6] J. H. Hayes and J. Offutt. Recognizing authors: an examination of the consistent programmer hypothesis. Software Testing, Verification and Reliability, 20(4):329– 356, 2010.

[7] I. Krsul and E. H. Spafford. Authorship analysis: Identifying the author of a program. Computers & Security, 16(3):233–257, 1997.

[8] S. Burrows, A. L. Uitdenbogerd, and A. Turpin. Application of information retrieval techniques for source code authorshipattribution. InDatabaseSystemsforAdvanced Applications, pages 699–713. Springer, 2009.

[9] 2014 April URL http://sideeffect.kr/popularconvention

[10] Zhang Xuegong. On the statistical learning theory and support vector machine [J]. Automation journal, 2000, 26 (1).

[11] N. Rosenblum, X. Zhu, and B. Miller. Who wrote this code? Identifying the authors of program binaries. Computer Security–ESORICS 2011, pages 172–189, 2011.

[12] M. Shevertalov, J. Kothari, E. Stehle, and S. Mancoridis. On the use of discretized source code metrics for author identification. In Search Based Software Engineering, 2009 1st International Symposium on, pages 69–78. IEEE, 2009.

[13] G. Frantzeskou, S. MacDonell, E. Stamatatos, and S. Gritzalis. Examining the significance of high-level programming features in source code author classification. Journal of Systems and Software, 81(3):447–460,2008